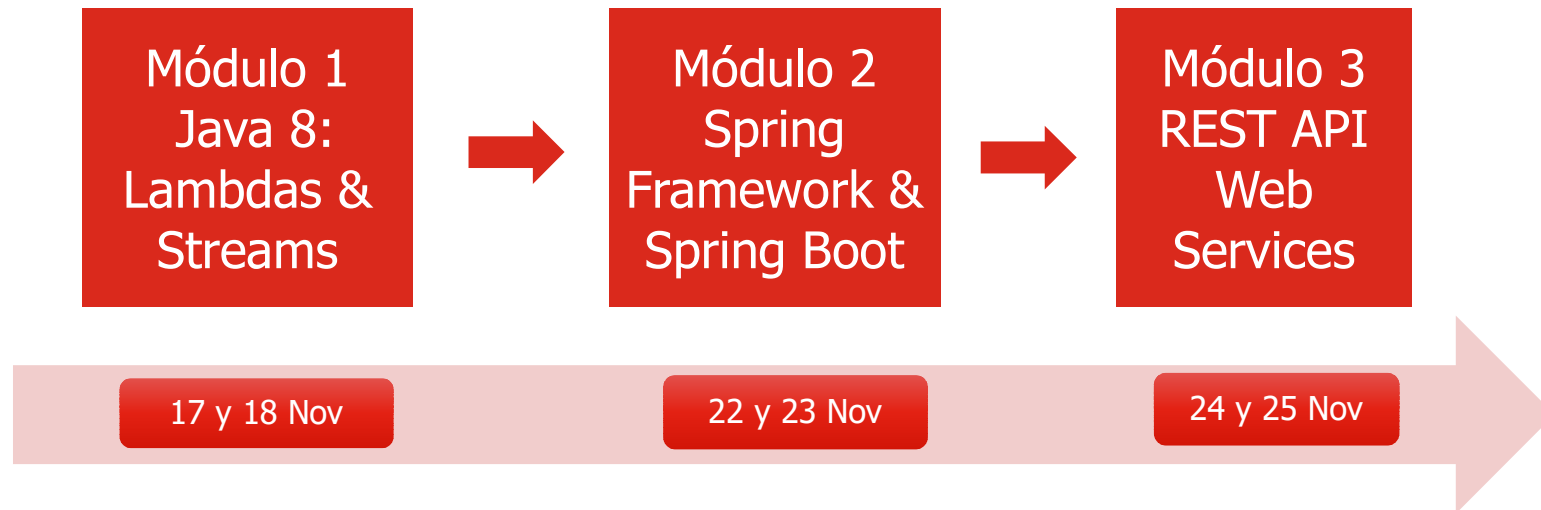


Programa Juniors Backend GFT



Formador: Ezequiel Llarena Borges

Bloques principales de contenidos

Módulo 1
Java 8:
Lambdas &
Streams

**Java
Lambdas
Expressions**

**Streams
API**

**Java
Developer
Roadmap**

Objetivos del curso



Proporcionar una introducción a las expresiones Lambdas y Streams de Java 8:

- Utilizar expresiones Lambdas Java 8
- Utilizar Java 8 Streams API
- Tener una visión general del Java Developer Roadmap

(Índice)

1. Java SE 8
2. Lambda Expressions
3. Stream API
4. Date API
5. Java Developer Roadmap

1. Java SE 8

Evolución de JAVA

Sun Microsystems

Lanzamiento	Año
JDK Beta	1994
JDK 1.0	1996
JDK 1.1	1997
J2SE 1.2	1998
J2SE 1.3	2000
J2SE 1.4	2002
J2SE 5.0	2004
Java SE 6	2006



Evolución de JAVA

2010 - Sun Microsystem comprado por Oracle

Cambio principal introducido por Oracle:

- Plataforma abierta para Java: **Open JDK**
- Plataforma comercial uso empresarial: **Oracle Java**



Evolución de JAVA

Oracle Inc

Lanzamiento	Año	
Java SE 7	2011	
Java SE 8	2014	LTS
Java SE 9	2017	
Java SE 10	2018	
Java SE 11	2018	LTS
Java SE 12	2019	



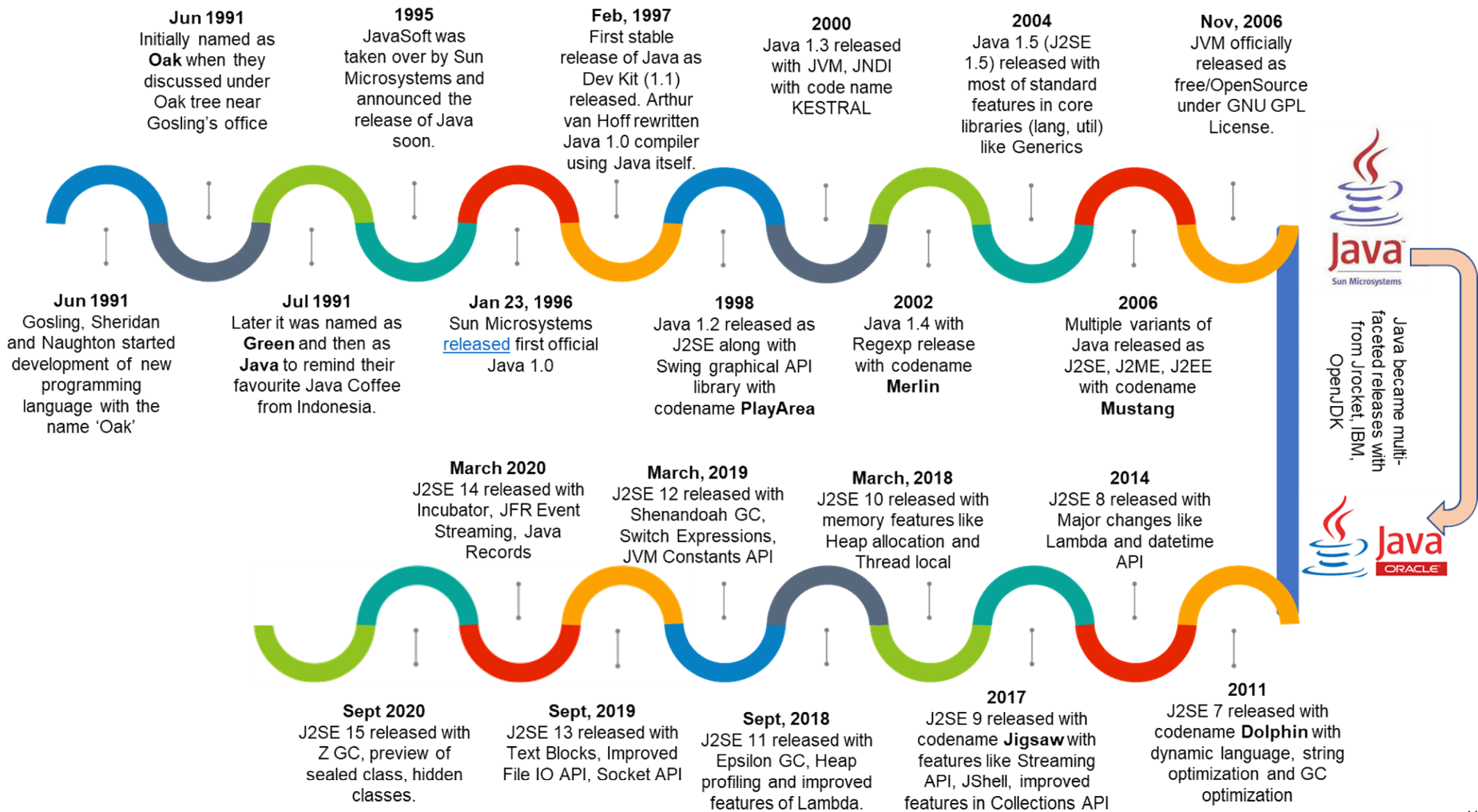
JAVA

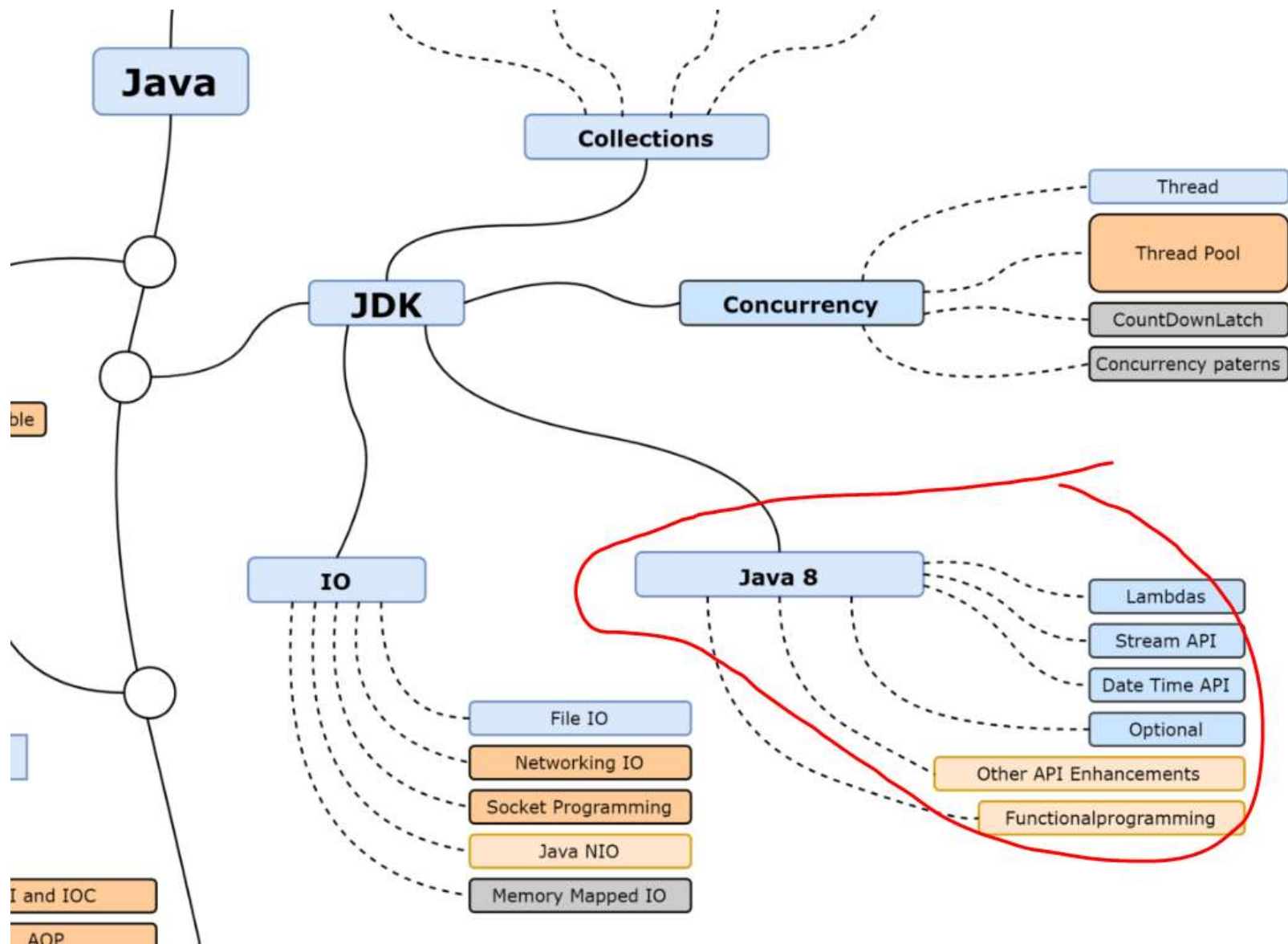
Oracle JDK

2011	Java 7	<ul style="list-style-type: none">• Soporte colecciones• Cierre recursos• Inferencia tipos• NIO 2.0	<ul style="list-style-type: none">• Socket DDP• Concurrencia• Internalización
2014	Java 8	<ul style="list-style-type: none">• Lambdas• Streams• Nuevo Api Time	<ul style="list-style-type: none">• Métodos por defecto• Interfaces funcionales• Http2 Api
2017	Java 9	<ul style="list-style-type: none">• Modularidad• Json Api• Optimización JVM	<ul style="list-style-type: none">• Java Shell• Http2 Api
2018	Java 10	<ul style="list-style-type: none">• Variables inferidas• Extensión CDS• Extensión Unicode	<ul style="list-style-type: none">• Versionamiento• Mejoras GC• Tratamiento hilos
2018	Java 11	<ul style="list-style-type: none">• Valhalla: Mejorar tratamiento de datos• Loom: Fibras o hilos más ligeros• Panama: Facilitar trabajo con código nativo• ZGC: Crear recolector para GB y TB• Amber: Literales strings raw	

Oracle JDK vs Open JDK

Comparativa	
Licencias	Oracle JDK las empresas o particulares no podrán tener acceso a los updates de la plataforma después de Enero de 2019 a menos que compres una licencia con Oracle. En el caso de Open JDK el uso es libre puedes crear una aplicación comercial o una sin animos de lucro.
Performance	No hay diferencia significativa entre ambas plataformas. La construcción de Oracle JDK está basada en Open JDK - En el caso de los bugs en Oracle JDK son corregidos inmediatamente y distribuidos a sus clientes de pago, en Open JDK debemos esperar a tener las correcciones (aportaciones de la comunidad Open Source).
Popularidad	Oracle JDK mantenida 100% por Oracle Corporation. Open JDK desarrollada por comunidad Java y una parte muy pequeña por Oracle y otras empresas como Red Hat, IBM, Azul systems, Apple Inc, SAP





Java 8

Métodos default en interfaces

Declaración de comportamiento por defecto en método de interfaz

Interfaces funcionales

`java.util.function`
`@FunctionalInterface`

Lambda

Expresiones lambdas

Stream API

Permite realizar distintas operaciones funcionales sobre streams
`java.util.Stream`

Date API

`java.time` API
tratamiento de fechas, tiempos, instantes y duraciones

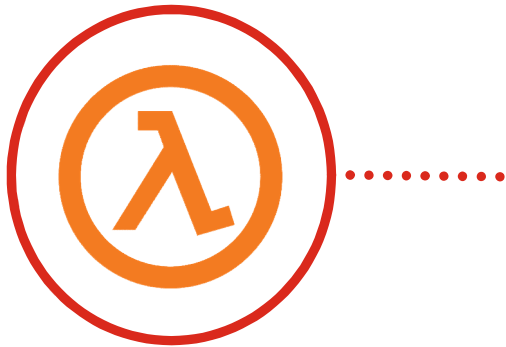
Excepciones y métodos nuevos en clases uso común

Integración con servicios Cloud

2. Lambda Expressions

Expresión Lambda de Java

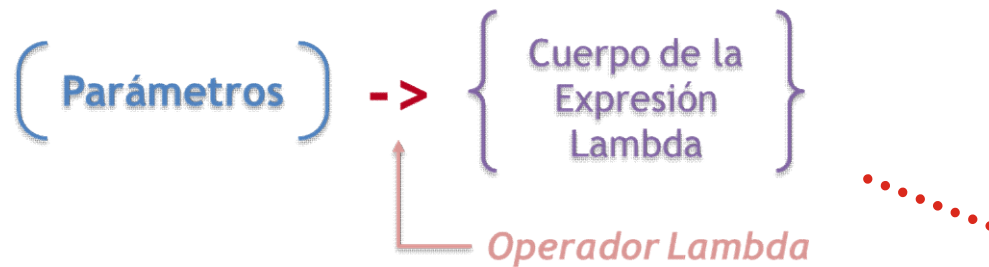
¿Qué es?



- **Función anónima**
- Método abstracto (declarado en una interfaz pero no implementado)
- Permite
 - tratar la funcionalidad o comportamiento como argumento de un método (el código como datos)
 - expresar instancias de clases de un solo método de forma más compacta
- Programación Imperativa vs Programación Funcional
- Soporte a la **programación funcional** junto con la API Stream

Expresión Lambda de Java

Sintaxis



- **(argumentos) -> { cuerpo }**
- Ejemplos:
 - (arg1, arg2,...) -> {cuerpo}
 - (int x) -> {cuerpo}
 - (int x, int y,...) -> {cuerpo}
 - x -> {cuerpo}
 - () -> {cuerpo}

Expresión Lambda de Java

Sintaxis en los argumentos

Declaración Explícita de Argumentos	Argumentos Inferidos
(String str) -> {cuerpo}	(x) -> {cuerpo}
String str -> {cuerpo}	x -> {cuerpo}
(int a, long b) -> {cuerpo}	(a, b) -> {cuerpo}
() -> {cuerpo}	(a, b, int c) -> {cuerpo}
En ambos casos: si hay más de un argumento, entre paréntesis y separados por coma	

Expresión Lambda de Java

Sintaxis en el cuerpo

Entre llaves obligatorio	Opcional
Cuando devuelve más de un valor o el cuerpo tiene más de una instrucción	Cuando devuelve un sólo valor, aunque el compilador no muestra error si se ponen llaves
<code>(int a, int b) -> { return a + b; }</code>	<code>() -> 10</code>
<code>() -> { return 3.1415; }</code>	<code>email -> System.out.println(email)</code>
	<code>s -> s.length() > 0</code>
<code>p -> { return p.getGender() == Person.Sex.MALE && p.getAge() >= 18 && p.getAge() <= 25; }</code>	<code>p -> p.getGender() == Person.Sex.MALE && p.getAge() >= 18 && p.getAge() <= 25</code>

Interfaz Funcional

¿Qué es?

- Interfaz que tiene **un solo método abstracto**
- Debe implementar sus métodos dentro la misma interfaz (esto no se podía hacer en versiones anteriores), para esto se debe anteponer la palabra reservada *default* al inicio de la declaración del método.
- **@FunctionalInterface** (opcional)

Interfaz Funcional

Ejemplo

```
@FunctionalInterface
public interface IFuncionLambda {
    public void calcular(int a, int b);
}
```

```
public class TestLambdas {
    public static void main(String[] args) {

        int x = 10;
        int y = 5;

        IFuncionLambda iflambda =

            (a, b) -> { System.out.println(a + b);    };

        iflambda.calcular(x, y);

    }
}
```

Expresión Lambda de Java

Tipos

Predicados	Funciones	Proveedores	Consumidor
Predicate<T>	Function<T,R>	Supplier<T>	Consumer<T>
expresiones que reciben un argumento y devuelven un valor lógico	reciben un argumento y devuelven un resultado	no tiene parámetros de entrada, pero si devuelven un resultado	un sólo argumento de entrada y no devuelven ningún valor

3. Stream API

API Stream

¿Qué es?

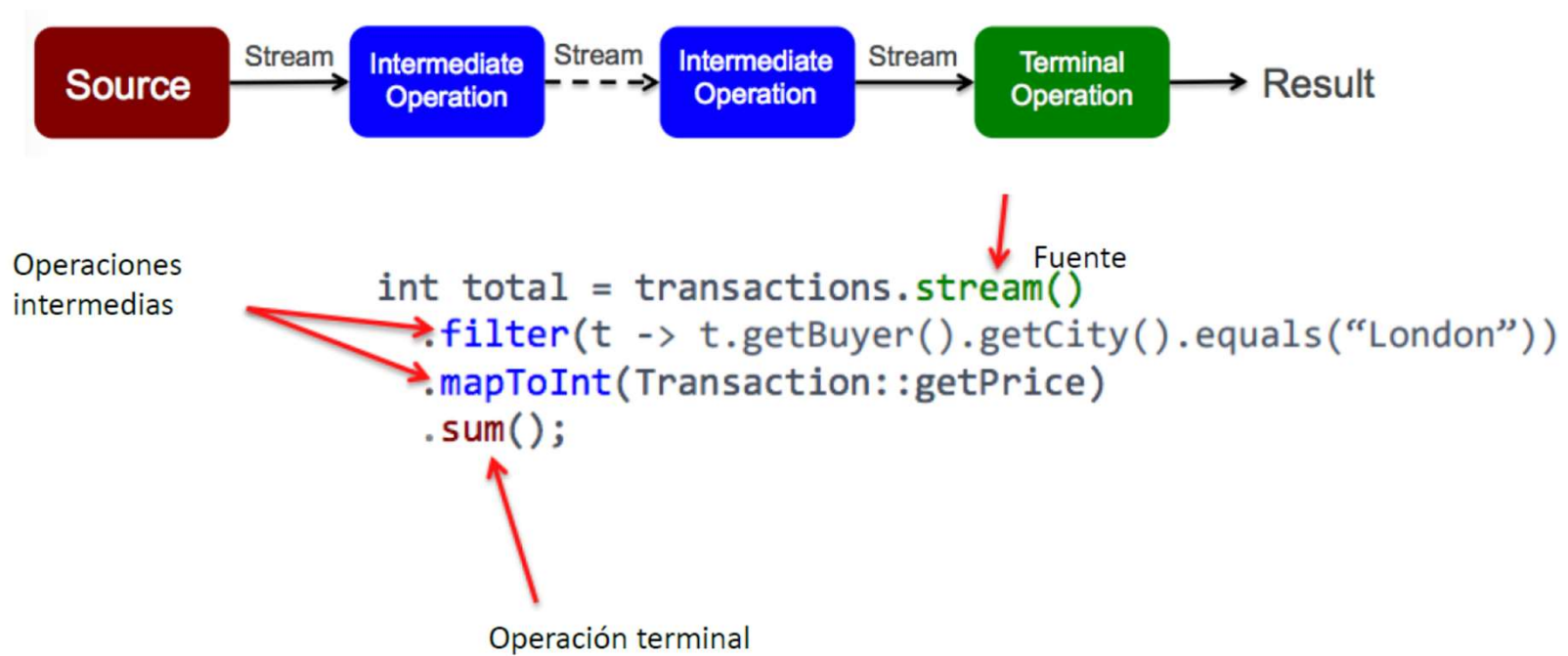
- Biblioteca de clases para realizar operaciones con **streams** de Java
- API Stream permite escribir código eficiente, limpio y conciso

`stream()`

- Un array es una colección de datos. Por lo que es capaz de crear un stream
- Proporciona un stream secuencial
- métodos sobrecargado para diferentes tipos
 - `double`, `int`, `long`, `Object`

Stream

Descripción



Stream Interface

Intermediate Operations

Un Stream proporciona una secuencia de elementos

- Soportan operaciones de agregado secuenciales o en paralelo

La mayoría de las operaciones requieren un parámetro que describe su comportamiento

- En general utilizando expresiones lambda
- La mayoría no modifican el stream (Non-Interfering)
- En general son sin estado

Stream Interface

Operaciones Intermedias: filtrado y mapeo

`distinct()`

- Regresa un flujo sin elementos duplicados

`filter(Predicado p)`

- Regresa un flujo con aquellos elementos que cumplen el predicado

`map(Function p)`

- Regresa un stream en donde la función proporcionada es aplicada a cada uno de los elementos del stream

`mapToInt(), mapToDouble(), mapToLong()`

- Funcionan como `map()` pero producen streams de primitivas es vez de objetos

Stream Interface

Operaciones Terminales

Finaliza la tubería de operaciones sobre el stream

Únicamente hasta este punto se realizan los procesamientos

- Esto permite la optimización de la tubería
- Evaluación perezosa (Lazy)
- Operaciones de fusión o de fusión
- Eliminación de operaciones redundantes
- Ejecución en paralelo

Genera un resultado explícito o un efecto secundario

Stream Interface

Operaciones Terminales: elementos coincidentes

`findFirst()`

- La primer coincidencia

`findAny()`

- Trabaja igual que `findFirst()`, pero actúa sobre un stream paralelo
- `boolean allMatch(Predicate p)`
 - Si todos los elementos del stream coinciden utilizando el Predicado p
- `boolean anyMatch(Predicate p)`
 - Si al menos uno de los elementos coincide con la condición del predicado p
- `boolean noneMatch(Predicate p)`
 - ninguno de los elementos coincide de acuerdo al predicado

Stream Interface

Operaciones Terminales: resultados numéricos

`count()`

- Regresa el numero de elementos que hay en el stream

`max (Comparator c)`

- El elemento con el máximo valor que está dentro del stream en base al comparador

`min(Comparator c)`

- El elemento con el valor más pequeño que está dentro del stream en base al comparador
- Regresa un Optional, si el stream esta vacío

`average()`

- Regresa la media aritmética del stream
- Si el stream está vacío. regresa un Optional

`sum()`

- Regresa la suma de los elementos del stream

Stream Interface

Operaciones Terminales: iteración

`forEach(Consumer c)`

- Realiza una acción por cada elemento del stream

`forEachOrdered(Consumer c)`

- Funciona igual que `forEach`, pero asegura que el orden de los elementos (Si hay elementos) es respetado cuando se utiliza por un stream paralelo

Utilízalo con precaución

- Alienta el estilo de programación no-funcional (imperativo)

Stream Interface

Clase Optional

- Ciertas situaciones en Java devuelven como resultado un valor null: referencia a un objeto que no ha sido inicializado

Las operaciones terminales como `min()` , `max()` podrían no poder regresar un resultado directo

- Sup Ciertas situaciones en Java regresan como resultado un valor null
- El cual referencia a un objeto que no ha sido inicializado

Optional

- Contenedor para una referencia de objeto (null o objeto real)
- Considerelo como un stream con 0 o 1 elemento
- Garantizado que la referencia Optional regresada no es null

4. Date API

API Date

Java 8 Date/Time API

- Se basa en la biblioteca de Java Joda Time
- API muy clara, concisa y fácil de entender
- Simplifica drásticamente el procesamiento de la fecha y la hora y corrige muchas deficiencias de la antigua biblioteca de fechas
- Flexibilidad: múltiples representaciones del tiempo.: `java.util.time.*`
- Immutability and Thread-Safety
 - Todas las representaciones de tiempo en la API de fecha y hora de Java 8 son inmutables y, por tanto, seguras para los hilos.
 - Todos los métodos de mutación devuelven una nueva copia en lugar de modificar el estado del objeto original.
 - Las antiguas clases como `java.util.Date` no eran seguras para los hilos y podían introducir errores de concurrencia muy sutiles.

API Date

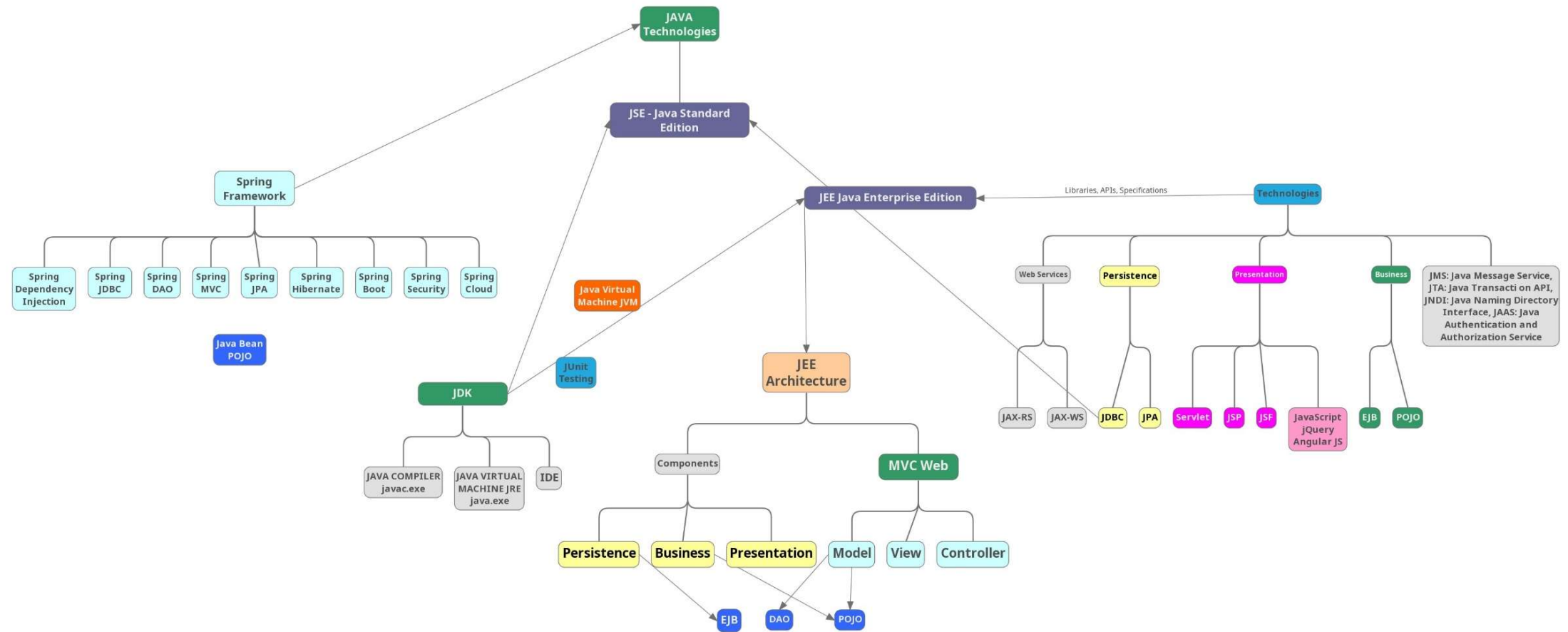
Java 8 Date/Time API

- Flexibilidad: múltiples representaciones del tiempo.: **java.util.time.***
 - *Instant* – represents a point in time (timestamp)
 - *LocalDate* – represents a date (year, month, day)
 - *LocalDateTime* – same as *LocalDate*, but includes time with nanosecond precision
 - *OffsetDateTime* – same as *LocalDateTime*, but with time zone offset
 - *LocalTime* – time with nanosecond precision and without date information
 - *ZonedDateTime* – same as *OffsetDateTime*, but includes a time zone ID
 - *OffsetLocalTime* – same as *LocalTime*, but with time zone offset
 - *MonthDay* – month and day, without year or time
 - *YearMonth* – month and year, without day or time
 - *Duration* – amount of time represented in seconds, minutes and hours. Has nanosecond precision
 - *Period* – amount of time represented in days, months and years

API Date

	Old API	New API
Getting current time	<code>Date now = new Date();</code>	<code>ZonedDateTime now = ZonedDateTime.now();</code>
Representing specific time	<code>Date birthDay = new GregorianCalendar(1990, Calendar.DECEMBER, 15).getTime();</code>	<code>LocalDate birthDay = LocalDate.of(1990, Month.DECEMBER, 15);</code>
Extracting specific fields	<code>int month = new GregorianCalendar().get(Calendar.MONTH);</code>	<code>Adding and subtracting time Month month = LocalDateTime.now().getMonth();</code>
Adding and subtracting time	<code>GregorianCalendar calendar = new GregorianCalendar(); calendar.add(Calendar.HOUR_OF_DAY, -5); Date fiveHoursBefore = calendar.getTime();</code>	<code>LocalDateTime fiveHoursBefore = LocalDateTime.now().minusHours(5);</code>
Number of days in a month	<code>Calendar calendar = new GregorianCalendar(1990, Calendar.FEBRUARY, 20); int daysInMonth = calendar.getActualMaximum(Calendar.DAY_OF _MONTH);</code>	<code>int daysInMonth = YearMonth.of(1990, 2).lengthOfMonth();</code>

5. Java Developer Roadmap



Bibliografía y webgrafía



Webs de referencia

<https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>

<http://stackoverflow.com/>

<http://www.codecademy.com/>



DEBATE

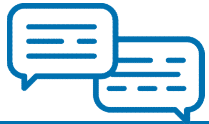
Programación Funcional

OBJECTIVE

Relacionar y consolidar los conceptos de las expresiones lambda, Stream y la programación funcional.

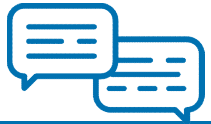
INSTRUCTIONS

1. Debatir en grupo qué entiendes por Programación Funcional
2. Poner en común todas las aportaciones y sacar conclusiones entre todos.



15 min

REFLEXIONES



15 min